A Parallel Primal-Dual Simplex Algorithm

Diego Klabjan *
Ellis L. Johnson
George L. Nemhauser

Email: diego,ellis.johnson,george.nemhauser@isye.gatech.edu

School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, GA 30332-0205

May 2, 2000

Abstract

Recently, the primal-dual simplex method has been used to solve linear programs with a large number of columns. We present a parallel primal-dual simplex algorithm that is capable of solving linear programs with at least an order of magnitude more columns than the previous work. The algorithm repeatedly solves several linear programs in parallel and combines the dual solutions to obtain a new dual feasible solution. The primal part of the algorithm involves a new randomized pricing strategy. We tested the algorithm on instances with thousands of rows and tens of millions of columns. For example, an instance with 1700 rows and 45 million columns was solved in about 2 hours on 12 processors.

Keywords: Programming/linear/algorithms,large scale systems.

1 Introduction

This paper presents a parallel primal-dual simplex algorithm that is capable of solving linear programs with thousands of rows and millions of columns. For example, an instance with 1700 rows and 45 million columns was solved in about 2 hours on 12 processors. The largest instance we solved has 25,000 rows and 30 million columns. Such linear programs arise, for example, in airline crew scheduling and several other applications as relaxations of set partitioning problems, Barnhart et al. (1998).

*Current address: Department of Mechanical and Industrial Engineering, University of Illinois at Urbana-Champaign, Urbana, IL, 61801

Primal-dual algorithms appear to have originated with Dantzig, Ford and Fulkerson (1956) and have been applied to many combinatorial problems such as network flows Ahuja, Magnanti and Orlin (1993), and matching, Edmonds (1965). Recently, Hu (1996) and Hu and Johnson (1999) have developed a primal-dual simplex algorithm that is designed to solve LPs with a very large number of columns. The primal-dual simplex iterates between solving primal subproblems over a restricted number of columns and dual steps. In this paper, we parallelize the dual step of the Hu-Johnson algorithm and obtain significant speedups. Since the number of dual steps is small, information is passed infrequently, so it is efficient to partition the columns among several computers, i.e. to use a distributed memory system.

Bixby and Martin (1995) present a dual simplex algorithm with parallel pricing. However, their approach requires a shared memory system to achieve good speedups since pricing is done over all columns after every simplex iteration. Bixby and Martin cite other literature on parallel simplex algorithms, none of which is related to our work.

In Section 2, we review the primal-dual algorithm. Section 3 presents the parallel primal-dual algorithm. Section 4 gives the computation results.

2 Primal-Dual Algorithm

We consider the LP

$$\min\{cx : Ax = b, x \geq 0\}\,, \tag{P}$$

where $A \in \mathbb{Q}^{n\times m}, b \in \mathbb{Q}^m, x \in \mathbb{Q}^n$ and $\mathbb{Q}^k$ is the set of k-dimensional rational numbers.

The dual is

$$\max\{b\pi : \pi A \leq c\}\,. \tag{D}$$

The primal-dual algorithm is efficient only if the number of columns is much larger than the number of rows, so we assume that $n \gg m$.

A primal subproblem is a problem with only a subset of columns from the constraint matrix A and corresponding objective coefficients. The reduced cost of column j with respect to a dual vector π is $rc_j^\pi = c_j - \pi A^j$, where A^j is the jth column of A.

We start with a statement of the primal-dual algorithm 1 from Hu (1996).

The algorithm maintains a dual feasible solution π. At each iteration the algorithm produces a dual vector ρ which is not necessarily dual feasible. The two dual vectors are then combined into a new dual feasible vector that gives the largest improvement to the dual objective. The dual vector is used to select columns for the next primal subproblem. The procedure is then iterated.

3 Parallel Primal-Dual Algorithm

3.1 Algorithm

The key idea of parallelization is to form several primal subproblems that are solved in parallel. The new dual feasible vector is a convex combination of optimal dual vectors

Algorithm 1 The Primal-Dual Algorithm

1: A dual feasible solution π and a primal feasible subproblem are given.
2: Solve the subproblem and let ρ be a dual optimal solution and x a primal optimal solution.
3: If $b\pi = cx$, then (x, π) are optimal solutions. If ρ is dual feasible, then (x, ρ) are optimal solutions.
4: Find an $\alpha \in \mathbb{Q}, 0 < \alpha < 1$ such that $(1-\alpha)\pi + \alpha\rho$ is dual feasible and α is maximum. Set $\pi = (1-\alpha)\pi + \alpha\rho$.
5: Remove all the columns from the subproblem except the basic ones. Add a set of columns with the lowest reduced costs rc_j^{π} to form a new subproblem.
6: Go to step 2.

arising from the subproblems and the current dual feasible solution. This can be seen as a natural way of parallelizing the primal-dual algorithm since each successive π is a convex combination of the initial π and all previous ρs.

Suppose we have p processors and each column of A is assigned with equal probability to a processor. A detailed description of the parallel primal-dual algorithm 2 follows.

Algorithm 2 The Parallel Primal-Dual Algorithm

1: A dual feasible solution π and p primal feasible subproblems $P_i, i = 1, \ldots, p$ are given.
2: **for** $i = 1$ to p **in parallel do**
3: Solve the subproblem P_i and let ρ_i be a dual optimal solution and x_i a primal optimal solution.
4: If $b\pi = cx_i$, then (x_i, π) are optimal solutions.
5: **end for**
6: Find $\alpha \in \mathbb{Q}_+^p, \sum_{i=1}^p \alpha_i \leq 1$ such that

$$\widetilde{\pi} = (1 - \sum_{i=1}^{p} \alpha_i)\pi + \sum_{i=1}^{p} \alpha_i \rho_i \tag{1}$$

is dual feasible and $b\widetilde{\pi}$ is maximum. Set $\pi = \widetilde{\pi}$.
7: If $b\pi = cx_i$ for some i, then (x_i, π) are optimal solutions.
8: **for** $i = 1$ to p **in parallel do**
9: Remove all the columns from the subproblem P_i except the basic columns. Using π and controlled randomization append new columns to P_i.
10: **end for**
11: Go to step 3.

The details for steps 1,6 and 9 are given below.

Forming Initial Subproblems (Step 1)

Since it is expensive to compute the reduced costs of a large number of columns, unless the initial dual solution π is thought to be 'good', we choose columns for the initial subproblems randomly. We add artificial variables with big costs to each subproblem to make them primal feasible. Once an artificial variable becomes nonbasic, we remove it permanently from the subproblem (step 3, or step 9 if the internal functionalities of the linear programming solver are not available).

On the other hand, if the initial π is thought to be 'good', then the controlled randomization procedure (step 9) described below is applied.

Combining Dual Solutions (Step 6)

In step 6 we find a convex combination of vectors $\pi, \rho_1, \ldots, \rho_p$ that yields a dual feasible vector and gives the largest increase in the dual objective value. Let $v = b\pi$ and $v_i = b\rho_i, i = 1, \ldots, p$. Note that $v_i \geq v$ for all i by weak duality. The dual feasibility constraints

$$(1 - \sum_{i=1}^{p} \alpha_i)\pi A + \sum_{i=1}^{p} \alpha_i \rho_i A \leq c$$

can be rewritten as

$$\sum_{i=1}^{p} \alpha_i (rc^{\pi} - rc^{\rho_i}) \leq rc^{\pi} .$$

Hence α can be obtained by solving the LP

$$\begin{aligned} \max \quad & \sum_{i=1}^{p} \alpha_i (v_i - v) \\ & \sum_{i=1}^{p} \alpha_i (rc^{\pi} - rc^{\rho_i}) \leq rc^{\pi} \\ & \sum_{i=1}^{p} \alpha_i \leq 1 \\ & \alpha \geq 0 . \end{aligned} \tag{2}$$

This is an LP with p variables and $n + 1$ constraints, where p is very small and n is very large. Its dual is

$$\begin{aligned} \min \quad & z + \sum_{j=1}^{n} rc_j^{\pi} y_j \\ & z + \sum_{j=1}^{n} (rc_j^{\pi} - rc_j^{\rho_i}) y_j \geq v_i - v, \quad i = 1, \ldots, p \\ & z \geq 0, \ y_1, \ldots, y_n \geq 0 . \end{aligned} \tag{3}$$

We solve (3) with a SPRINT approach due to Forrest, see Anbil, Johnson and Tanga (1992) and Bixby et al. (1992). We start with a subproblem consisting of the z column and some randomly chosen y columns. Then we iterate the following steps. Let α be an optimal dual vector. Delete all the columns from the subproblem except the basic columns and the z column. Add to the current subproblem a subset of columns of (3) with the smallest reduced cost based on α, and then solve it. If all the columns have nonnegative reduced cost, then the solution is optimal.

If we compute reduced costs for (3) directly from the constraints in (2), then for a single column we would have to compute p reduced costs for a column in P, one for each ρ_i, and then compute their sum weighted with the dual vector α of (3). Instead we can compute the reduced cost rc_j more efficiently by rewriting

$$\begin{aligned} rc_j &= rc_j^{\pi} - \sum_{i=1}^{p} \alpha_i (rc_j^{\pi} - rc_j^{\rho_i}) \\ &= c_j - \sum_{k=1}^{m} ((1 - \sum_{i=1}^{p} \alpha_i)\pi_k + \sum_{i=1}^{p} \alpha_i \rho_k^i) a_{kj} \\ &= c_j - \widetilde{\alpha} A^j = rc_j^{\widetilde{\alpha}} , \end{aligned}$$

where $\widetilde{\alpha} \in \mathbb{Q}^m$ with $\widetilde{\alpha}_k = (1 - \sum_{i=1}^{p} \alpha_i)\pi_k + \sum_{i=1}^{p} \alpha_i \rho_k^i, k = 1 \ldots, m$. Hence at each iteration of SPRINT, before forming the new subproblem, we first compute $\widetilde{\alpha}$ and then the pricing for (3) is equivalent to the pricing for P with respect to $\widetilde{\alpha}$.

We now show that the stopping criteria in step 7 is also implied by the dual feasibility of the ρs. Note that the dual feasibility check could be also done after step 5 but that would be very costly to perform because of the huge number of columns.

Proposition 1. *If $\sum_{j=1}^{p} \alpha_j = 1$ in an optimal solution to (2) , then there is an i such that $b\pi = cx_i$. If ρ_i is dual feasible, then $b\pi = cx_i$.*

Proof. Let $v = b\pi$ and $v_j = cx_j$. Assume that $\sum_{j=1}^{p} \alpha_j = 1$. Then by duality and (1) $v = \sum_{j=1}^{p} \alpha_j v_j$ and hence $\sum_{j=1}^{p} \alpha_j (v_j - v) = 0$. Since $v_j - v \geq 0$ by weak duality and $\alpha_j \geq 0$, it follows that $\alpha_j (v_j - v) = 0$ for all j. Since there is an i such that $\alpha_i > 0$, $v = v_i$.

If ρ_i is dual feasible, then $\alpha_i = 1$ and $\alpha_j = 0$ for all $j \neq i$ is feasible to (2) and optimal by weak duality. Hence $v = v_i$. □

Choosing Columns for the Subproblems (Step 9)

In addition to needing columns with low reduced cost, it is important that subproblems receive a representative sample of columns from the original problem. This is achieved by a controlled randomization process based on the current dual solution.

The following parameters are used in assigning columns.

nSub: the expected number of columns given to a subproblem (this is determined empirically);

nSubLow: the number of common low reduced cost columns given to each subproblem in Case 1 below (to be discussed later in the section);

$nSub0$: the number of columns j with $rc_j^\pi = 0$;

$minRC$: if $nSub0 < nSubLow$, then $minRC > 0$ is a number such that the number of columns with reduced cost less than $minRC$ is $nSubLow$;

$maxRC$: if $rc_j^\pi > maxRC$, then column j is not considered as a candidate for a subproblem;

$nSubHigh$: the number of columns j with $rc_j^\pi \le maxRC$.

To compute $maxRC$, first note that we need $p \cdot nSub$ columns. Also we know that there are $nSubLow$ columns with reduced cost below $minRC$. Thus assuming at the first iteration that the columns have been uniformly distributed among the processors, we have

$$maxRC = \frac{p \cdot minRC \cdot nSub}{nSubLow} .$$

We compute $nSubHigh$ only at the first iteration. In subsequent iterations we retain the value of $nSubHigh$ and adjust $maxRC$ accordingly.

We consider 3 cases.

Case 1) **nSub0 < nSubLow**

All columns j with $rc_j^\pi < minRC$ are given to each subproblem. Every column j with $minRC \le rc_j^\pi \le maxRC$ is a candidate for a subproblem. We select columns using the idea that the lower reduced cost columns should have a higher probability of being selected.

Let p_j be the probability that a column j from the initial constraint matrix A is added to subproblem i. Since there is no reason to distinguish between subproblems, the probability does not depend on the subproblem index i. Clearly p_j should be a nonincreasing function of the reduced cost rc_j^π and

$$\sum_{\substack{j=1 \\ minRC \le rc_j^\pi \le maxRC}}^{n} p_j = nSub - nSubLow .$$

In order to emphasize the reduced cost we choose the rapidly decreasing function $\exp(-\tau x^2)$. The value τ is determined by

$$f(\tau) = \sum_{\substack{j=1 \\ minRC \le rc_j^\pi \le maxRC}}^{n} \exp(-\tau (rc_j^\pi)^2) = nSub - nSubLow . \tag{4}$$

Because n is so large, it is too expensive to evaluate $f(\tau)$ or its derivative. Instead we approximate the value of τ using a 'bucket' approach.

Let N be the number of buckets (we use $N = 100 \ll n$). For $j = 0, \ldots, N-1$ define the jth bucket to contain columns

$$S_j = \{i : j \cdot a \le rc_i^\pi - minRC \le (j+1) \cdot a\} ,$$

where $a = \frac{maxRC - minRC}{N}$. Let $b_j = |S_j|$ and $R_j = \sum_{i \in S_j} rc_i^\pi / b_j$, the average reduced cost of the jth bucket. Let $\widetilde{f}(\tau) = \sum_{j=0}^{N-1} b_j \exp(-\tau R_j^2)$ and let τ^* be the solution to the equation $\widetilde{f}(\tau^*) = nSub - nSubLow$. It is easy to derive bounds

$$\frac{\ln(\frac{\sum_{j=0}^{N-1} b_j}{nSub - nSubLow})}{minRC^2} \leq \tau^* \leq \frac{\ln(\frac{\sum_{j=0}^{N-1} b_j}{nSub - nSubLow})}{maxRC^2} \,. \tag{5}$$

The function $\widetilde{f}$ is continuously differentiable and decreasing in the interval given by (5). If we use Newton's method with a starting point in the interval given by (5), the method converges to the solution of the equation $\widetilde{f}(\tau^*) = nSub - nSubLow$ (see e.g. Bertsekas (1995)). Newton's method is fast for small values of N and the optimal value τ^* is a good approximation to the solution of (4).

Case 2) $\mathbf{nSub0 \geq nSubLow, nSub0 \leq k \cdot nSub}$**, where** $\mathbf{k < 1}$ **is a parameter (we use** $\mathbf{k = \frac{1}{3}}$**).**

Replace $nSubLow$ by $nSub0$ and apply the procedure of Case 1.

Case 3) $\mathbf{nSub0 > \max(nSubLow, k \cdot nSub)}$

Since there are so many columns with $rc_j^\pi = 0$, we assign them randomly to the subproblems. Empirically, we obtained good results if the expected number of columns given to each subproblem is $\frac{nSub}{r}$ where

$$r = \max\{1, \frac{nSubHigh}{nSub0}, \frac{nSub0^2}{nSubLow \cdot nSub}\} \,.$$

The remaining columns are assigned by controlled randomization as in Case 1.

Pricing Heuristic

It remains to be shown how to efficiently find a reduced cost value such that the number of columns with the reduced cost smaller than that value is a given number k. This is required in both step 9 and the SPRINT algorithm used to solve (3). Using sorting terminology, we want to find a rank k element among reduced cost values. Since for our applications we have only an estimate of k, we want to find a reduced cost value that yields approximately k columns.

Other approaches are considered in Anbil, Johnson and Tanga (1992), Hu (1996) and Bixby et al. (1992). Bader and JaJa (1996) describe an algorithm for finding a median in parallel, but it is likely to be too slow for our needs since it is targeted to find exactly the rank k element.

Our approach relies on distributing the columns randomly. Assume that S_i is the set of reduced cost values at processor i. For simplicity of notation let d_j be the reduced cost of column j (instead of $rc_j^{(\cdot)}$), i.e. $S_i = \{d_1, \ldots, d_{n_i}\}$. Our goal is to find the kth smallest element in $\cup_{i=1}^p S_i$. For our applications k is always much smaller than n_i.

The following intuitive observation plays a key role. Let d_{m_i} be an element with rank $r = \lfloor \frac{k}{p} \rfloor$ in the sequence S_i and let $d = \min_{i=1,\ldots,p}\{d_{m_i}\}$. Let m be the number

of elements in $\cup_{i=1}^{p} S_i$ that are smaller than d. Since the numbers d_i are randomly distributed, m should be approximately $p \cdot r \approx k$. It is clear that $m \leq p \cdot r$. Experiments have shown the validity of the claim. Klabjan (1999) gives some theoretical support by proving that $r(p-1) \leq m$ as $n_i \to \infty$ for all i and $k \to \infty$.

So the task reduces to finding an element with rank r in S_i. Even here we do not sort the array S_i due to the possible large value of n_i. Any exact sequential median finding algorithm is likely to be too slow and too 'exact' since we are looking only for an approximation of a rank r element. Since k is typically a small number, so is r.

For simplicity we write n, S instead of n_i, S_i. Let s and $\tilde{r}, \tilde{r} \leq s$, be two integers. Suppose that we choose s elements uniformly at random from S and we denote them as $\widehat{S}$. Let $\tilde{d}$ be the element with rank $\tilde{r}$ in $\widehat{S}$. The following theorem forms the basis for our heuristic.

Theorem 1 (Klabjan (1999)**).** *If all elements in S are different, then*

$$\mathrm{E} = \mathrm{E}(|\{d_i : d_i \leq \tilde{d}\}|) = \frac{\tilde{r}(n+1)}{s+1} .$$

Since we want the sample size s to be as small as possible, we choose $\tilde{r} = 1$. Hence $s = \lceil n/r \rceil$. For our instances n ranged in millions and r was always bigger than 500. For example, if $r = 500$ and $n = 2 \cdot 10^6$, the sampling size is 40,000.

Note that in step 9 of the parallel primal-dual algorithm, we have to find elements of rank $nSubLow$ and $nSubHigh$. We need to obtain samples just once since we can use them for both computations.

3.2 Finite Convergence

We prove the convergence of the algorithm under the assumption of primal nondegeneracy.

Theorem 2. *If problem P is primal nondegenerate and at each iteration all the 0 reduced cost columns based on π are added to each subproblem, then the parallel primal-dual algorithm terminates finitely.*

Proof. Suppose the algorithm is in the kth major iteration after step 7. Denote by v_i^k the optimal value of subproblem i. Consider the LP (2) and its dual, and let $(\alpha;\ z, y)$ be optimal solutions. Since the stopping criteria in steps 4 and 7 are not satisfied, by Proposition 1, $\sum_{i=1}^{p} \alpha_i < 1$ and $v = b\pi < v_i^k$ for all $i = 1, \ldots, p$.

Suppose that $\alpha = 0$. Since $v < v_i^k$, this is the only feasible solution to (2). Therefore $conv\{\pi, \rho_1, \ldots, \rho_p\}$ is the singleton π, implying that $\pi = \rho_i$ for all $i = 1, \ldots, p$. Since $\rho_1 = \pi$ is dual feasible and by Proposition 1, the stopping criteria in step 7 is fulfilled, there is a contradiction. Thus $\alpha \neq 0$ and the optimal value to (3) is positive.

Since $\sum_{i=1}^{p} \alpha_i < 1$, by complementary slackness $z = 0$. The optimal value to (3) is positive and hence $\sum_{j=1}^{n} rc_j^{\pi} y_j > 0$. Since $rc_j^{\pi} y_j \geq 0$ for all $j = 1, \ldots, n$, there is an index $j_0, 1 \leq j_0 \leq n$ such that $y_{j_0} > 0$ and $rc_{j_0}^{\pi} > 0$. By complementary slackness $\sum_{i=1}^{p} \alpha_i (rc_{j_0}^{\pi} - rc_{j_0}^{\rho_i}) = rc_{j_0}^{\pi}$. Since $\sum_{i=1}^{p} \alpha_i < 1$ and $rc_{j_0}^{\pi} > 0$, it easily follows that there is an index $i_0, 1 \leq i_0 \leq p$ such that $rc_{j_0}^{\rho_{i_0}} < 0$.

Consider now the column j_0 of P. Since in (2) the row j_0 is at equality, it follows that $rc^{\widetilde{\pi}}_{j_0} = 0$. By assumption the column is appended to each subproblem, hence to the subproblem i_0. By the nondegeneracy assumption it follows that $v^{k+1}_{i_0} < v^k_{i_0}$.

Consider $o_k = \sum_{i=1}^{p} v^k_i$. Since $v^k_i \geq v^{k+1}_i$ for all $i = 1, \ldots, p$ and $v^k_{i_0} > v^{k+1}_{i_0}$, it follows that $o_k > o_{k+1}$ for all k. Since there are only finitely many subproblems, the number of different values of v^k_i for all $i = 1, \ldots, p$ and k is finite. The claim now follows due to the finite number of different values in the o sequence and the monotonicity property $o_k > o_{k+1}$. □

Note that the dual objective value increases at each iteration regardless of degeneracy, however the finiteness argument does not follow from the nondegeneracy assumption.

4 Computational Experiments

4.1 Computing Environment

All computational experiments were performed on a cluster of machines comprised of 48 300MHz Intel Dual Pentium IIs, resulting in 96 processors available for parallel program execution. All machines are linked via 100 MB point-to-point Ethernet switched via a Cisco 5500 network switch. Each node has 512MBytes of main memory.

The operating system used was Sun Solaris x86, version 2.5.1, which offers facilities for parallel computing like remote shell (rsh commands), global file system support via NFS, and parallel computing libraries like MPI or PVM. The cluster is representative of typical machines of this type, in its relatively slow internode communications and its good cost/performance ratio vs. specialized parallel machines like the CM-5, the Intel Paragon, or the IBM SP-2 machines.

The parallel implementation uses the MPI message passing interface MPI, MPICH implementation version 1.0, developed at Argonne National Labs. The MPI message passing standard is widely used in the parallel computing community. It offers facilities for creating parallel programs to run across cluster machines and for exchanging information between processes using message passing procedures like broadcast, send, receive and others.

The linear programming solver used was CPLEX, CPLEX Optimization (1997), version 5.0.

4.2 Problem Instances

The instances are listed in Table 1. The set partitioning instances sp1, sp2, and sp3 are airline crew scheduling problems (see e.g. Klabjan (1999)). All of these instances may contain some duplicate columns but, because of the method of generation, no more than approximately 5% of the columns are duplicates. For this reason we do not remove duplicate columns.

The remaining 2 instances also are from Klabjan (1999). The problems have a substantial set partitioning portion. They occured in a new approach to solving the airline crew scheduling problem. The pr2 problem is particularly hard due to the high number of rows and primal degeneracy.

Problem Name	Number of Rows	Number of Columns
pr1	10677	17,045,897
pr2	13048	9,234,109
sp1	449	42,134,546
sp2	1742	45,952,785
sp3	3143	46,546,240

Table 1: Problem statistics

All instances have on average 10 nonzeros per column.

4.3 Implementation and Parameters

Because some problems are highly primal degenerate, the primal LP solver struggles to make progress. Therefore we globally perturb the right hand side and then we gradually decrease the perturbation. Precisely, we perturb a row $A_i x = b_i$ of P to a ranged row $b_i - \epsilon_1 \leq A_i x \leq b_i + \epsilon_2$, where ϵ_1, ϵ_2 are small random numbers. There is no reason to find an optimal solution to the perturbed problem, so we apply the parallel primal-dual method until

$$\frac{\min_{i=1,\dots,p}\{v_i\} - v}{\min_{i=1,\dots,p}\{v_i\}} > gap\,,$$

where *gap* is a constant. The gap is checked at step 7 of the algorithm. If gap is below *gap*, then the perturbation is reduced by a factor of 2, i.e. each ϵ_1, ϵ_2 becomes $\frac{\epsilon_1}{2}, \frac{\epsilon_2}{2}$. Once all of the epsilons drop below 10^{-6}, the perturbation is removed entirely. For our experiments we set $gap = 0.03$.

For set partitioning problems a starting dual feasible vector is $\pi = 0$ if all the cost coefficients are nonnegative. However there are many columns with 0 cost resulting in many columns with 0 reduced cost. Hence we perturb π by componentwise subtracting a small random number. This decreases the initial dual objective but the new π is not 'jammed' in a corner. For instances pr1 and pr2 we use the same idea, however π needs to be changed to accommodate the extra rows and variables (see Klabjan (1999)).

At the first iteration we do not usually have a warm start and we found that the dual simplex is much faster than the primal simplex. However, a warm start is available at each successive iteration since we keep the optimal basis from the previous iteration. Hence the primal simplex is applied.

Next we discuss the parameters *nSub* and *nSubLow*. Empirically we found that $nSubLow = \frac{nSub}{2.5}$ works best. We use this relationship in all of our experiments. Table 2 gives *nSub* for the parallel primal-dual algorithm and the number of columns used in the sequential version of the code. In each case the parameters have been determined empirically to be the subproblem sizes that have given the best result. They will be discussed further in Section 4.4.

Finally, we observed empirically that for some instances (sp1 and sp2) and at certain iterations the algorithm spends too much time solving subproblems. So we impose an upper bound of 30 minutes on the execution time of subproblems. This improves the

Problem Name	Subproblem Size ($nSub$)	Subproblem Size, Seq.
pr1	17,500	30,000
pr2	12,500	30,000
sp1	10,000	10,000
sp2	10,000	35,000
sp3	5,000	10,000

Table 2: Subproblem sizes

overall execution time. Note that quitting the optimization before reaching the optimality of subproblems does not affect the correctness of the algorithm.

We start SPRINT for solving (3) with 40,000 random columns and in each successive iteration we append 50,000 best reduced cost columns. Typically 3 iterations of SPRINT are required, the third one just confirming optimality. The execution time never exceeded 1 minute.

4.4 Results

Due to the large number of columns and a main memory of 512MBytes, we were not able to solve any problem on a single machine. We implemented a variant of the sequential primal-dual algorithm in which the columns are distributed across the machines. Only the pricing is carried out in parallel. We call it the primal-dual algorithm with parallel pricing. In essence, a true sequential primal-dual algorithm would differ only in pricing all the columns sequentially. Based on the assumption that pricing is a linear time task (which is the case for our pricing procedure), we estimated the execution times of a true sequential primal-dual implementation.

The gain of the parallelization of the primal-dual algorithm is twofold; one is the parallel pricing strategy, and the second is having multiple subproblems. The first one is addressed by the primal-dual algorithm with parallel pricing. We give the speedups in Table 3. As we can see the parallel pricing heuristic has a big impact when the overall execution time is small and the number of columns is big, i.e. the sp1 problem. For the remaining problems the parallel pricing does not have a dominant effect.

Computational results using the parameters from Table 2 are presented in Table 4. The parallelization gain of having multiple subproblems is significant for problems with a large number of rows and relatively small number of columns, i.e. the pr1 and pr2 problems. The speedup is relatively high for a small number of processors, up to 8, and vanishes at 20 processors.

So our main conclusion is that the speedups are significant when the number of processors is between 4 and 12. The execution times can be improved even further by using a pure shared memory machine. On average the communication time was 45 seconds per iteration. A shared memory computing model would definitely lead to improvements for the sp1 problem. Additional improvement in the execution time of the sp1 problem might result by allowing π to be 'slightly' infeasible. When solving LP (3), we could perform just two SPRINT iterations and then quit without proving optimality.

Problem Name	Number of Processors	Execution Time (secs.)
pr1	1	19,200
	8	17,140
pr2	1	81,000
	8	76,860
sp1	1	3,000
	8	1,100
sp2	1	24,000
	8	10,320
sp3	1	62,460
	8	50,160

Table 3: Effect of parallel pricing

Problem Name	Number of Processors	Speedup	Execution Time (secs.)	Number of Iterations
pr1	1	1.00	19,200	44
	6	1.77	10,800	37
	9	2.09	9,180	30
	12	2.05	9,360	29
pr2	1	1.00	81,000	89
	4	1.74	46,500	87
	8	2.05	39,420	73
	12	2.13	37,860	72
	16	2.23	36,300	68
	20	2.17	37,260	67
sp1	1	1.00	3,000	11
	8	3.33	900	8
	12	4.16	720	7
	16	4.19	715	7
sp2	1	1.00	24,000	70
	12	3.22	7,440	25
	16	4.00	6,000	23
	20	3.41	7,020	21
sp3	1	1.00	62,460	62
	12	2.30	27,120	57
	16	2.50	24,960	53
	20	2.51	24,840	50

Table 4: Computational results

A breakdown of execution times is shown in Table 5. For the harder problems (pr1, pr2 and sp3) more than 70% of the time is spent in solving the subproblems, however for the sp1 problem only 17% of the time is spent on subproblem solving. A better communication network would improve the times for step 9, but for the harder problems it would not have a significant impact.

Problem Name	Total Time	Step 3 Time	Steps 6,7 Time	Step 9 Time
pr1	9360	5183	2727	1150
pr2	37860	32689	3166	1005
sp1	720	127	190	251
sp2	7440	3823	689	1700
sp3	27120	19425	1887	3420

Table 5: The breakdown of execution times on 12 processors

Although we use the same subproblem size regardless of the number of processors in our experiments reported in Table 4, the subproblem size should depend on the number of processors. The smaller the number of processors, the bigger the subproblem size should be. For a large number of processors, solving many small subproblems is better than spending time on solving larger subproblems. For the pr1 problem the optimal subproblem sizes are 30,000, 25,000, 17,500 for $p = 6, 9, 12$, respectively.

We would like to point out that there are no major synchronization requirements for subproblem solutions. The execution times of subproblems differ by a small amount, the average being 15 seconds. When the execution time for a subproblem reached the upper time limit, all the subproblems achieved the time limit. This fact is not surprising since the structure of the subproblems is the same.

The largest problem we have solved so far has 30 million columns and 25,000 rows, Klabjan (1999). The execution time on 12 processors was 30 hours.

There are several open questions regarding an efficient implementation of a parallel primal-dual simplex algorithm. Subproblem size is a key question and the development of an adaptive strategy could lead to substantial improvements. To make subproblems even more different, columns with negative reduced cost based on ρ_i can be added to the subproblems. We made an initial attempt in this direction but more experimentation needs to be done.

5 Acknowledgments

This work was supported by NSF grant DMI-9700285 and United Airlines, who also provided data for the computational experiments. Intel Corporation funded the parallel computing environment and ILOG provided the linear programming solver used in computational experiments.

References

Ahuja, R., Magnanti, T. and Orlin, J. 1993. *Network Flows.* Prentice Hall.

Anbil, R., Johnson, E. and Tanga, R. 1992. A Global Approach to Crew Pairing Optimization. *IBM Systems Journal*, **31**, 71–78.

Bader, D. and JaJa, J. 1996. Practical Parallel Algorithms for Dynamic Data Redistribution, Median Finding, and Selection, *10th International Parallel Processing Symposium.*

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M. and Vance, P. 1998. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, **46**, 316–329.

Bertsekas, D. 1995. *Nonlinear Programming*, Athena Scientific, 79–90.

Bixby, R., Gregory, J., Lustig, I., Marsten, R. and Shanno, D. 1992. Very Large-scale Linear Programming: A Case Study in Combining Interior Point and Simplex Methods. *Operations Research*, **40**, 885–897.

Bixby, R. and Martin, A. 1995. Parallelizing the Dual Simplex Method, *Technical Report CRPC-TR95706*, Rice University.

CPLEX Optimization 1997. *Using the CPLEX Callable Library*, 5.0 edn, ILOG Inc.

Dantzig, G., Ford, L. and Fulkerson, D. 1956. A Primal-dual Algorithm for Linear Programs. In *Linear Inequalities and Related Systems.* H. Kuhn and A. Tucker (editors). Princeton University Press, 171–181.

Edmonds, J. 1965. Maximum Matching and a Polyhedron with 0-1 Vertices. *Journal of Research of the National Bureau of Standards*, **69B**, 125–130.

Hu, J. 1996. *Solving Linear Programs Using Primal-dual Subproblem Simplex Method and Quasi-explicit Matrices.* Ph.D. Dissertation, Georgia Institute of Technology.

Hu, J. and Johnson, E. 1999. Computational Results with a Primal-dual Subproblem Simplex Method. *Operations Research Letters*, **25**, 149–158.

Klabjan, D. 1999. *Topics in Airline Crew Scheduling and Large Scale Optimization.* Ph.D. Dissertation, Georgia Institute of Technology.